

Sparkfun Artemis Global Tracker - Treasure Game



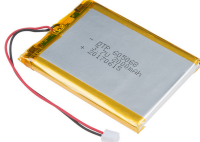
Jameson Koonce


The Story

The Artemis Global Tracker is a highly functional sensor device with professional grade pressure, humidity, and temperature sensing with the onboard TE MS8607 PHT sensor, and location data through the onboard u-blox ZOE-M8Q GNSS receiver. It can also be programmed with the Arduino IDE through the Artemis processor and transfer information through the Iridium Satellite Network due to the Iridium 9603N Short Burst Data modem. The goal of this project is to demonstrate the ease of working with this device, and the accuracy and potential of its onboard sensors, specifically the GNSS receiver.

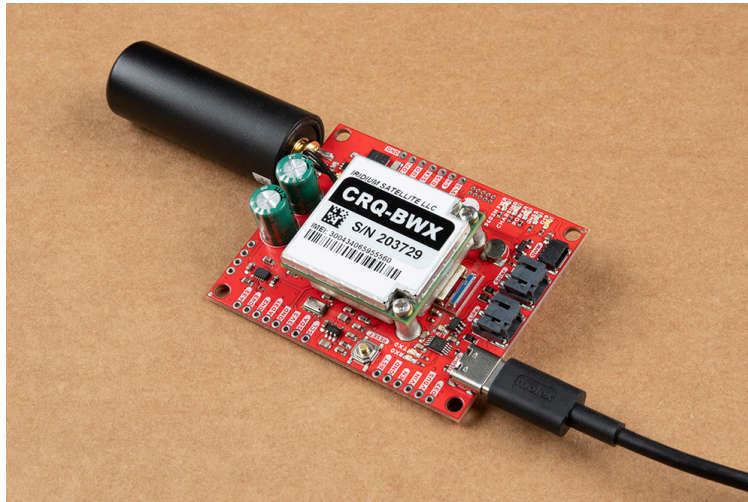
This project will use the Artemis Global Tracker (AGT) to create a “Treasure Game” in which the player begins the game, the AGT will randomly select a position in a 20 meter radius (designated the “treasure spot”), and the onboard LED will blink faster as the player moves toward the “treasure spot” - winning when they find it exactly. In this way, the onboard LED will lead the player to the “treasure spot”. This game will also transfer location information to a computer using the Iridium Satellite Network, meaning it could be played even where no WIFI or wireless data is accessible.

Here are the components required for this project:

Component	Description	Photo	Price
SparkFun Artemis Global Tracker	Tracker device powered by the Artemis Module		\$399.95
Iridium Passive Antenna	Antenna to connect to Iridium Satellite Network		\$64.95
Lithium Ion Battery	Battery to power AGT away from computer (computer may also be used with USB-C connection)		\$13.95

USB A to C Cable	Cable to connect AGT to computer to program through the Arduino IDE		\$5.50
------------------	---	--	--------

NOTE: To connect to the Iridium Satellite Network requires a monthly fee of \$17 and a purchase of credits, however, you can pick and choose the periods to access the network.



The Setup

The first step in the development setup is the software setup. If you have not programmed with the Arduino IDE before, SparkFun made a great introductory guide to setup with the Artemis module which can be [found here](#). It notes the need to add the Artemis module within the Arduino IDE boards manager using the following link:

```
https://raw.githubusercontent.com/sparkfun/Arduino_Apollo3/main/package_sparkfun_apollo3_index.json
```

You will also need to add the following libraries to the Arduino IDE: SparkFun u-blox GNSS Arduino Library, IridiumSBDi2c. If you want to make use of the PHT sensor, you will also need the SparkFun PHT MS8607 Arduino Library. These can all be found within the Arduino Library Manager.

To begin development, you will need to connect the Artemis Global Tracker to your device with the Arduino IDE using the USB-C cable, and select the designated board (RedBoard Artemis ATP).

The Logic

The code enables the GNSS module functionality, finds the current longitude and latitude of the AGT device, randomly generates a target distance and angle based on the current location, and begins blinking the onboard LED with intervals relating to the distance between the current location and target location. We will also mark the location of a 30 meter radius around the starting location using the “addGeofence” function from the SparkFun u-blox GNSS Arduino Library. If the device leaves this radius while playing the game, the player will lose. The treasure position is chosen by randomly selecting a distance between 15 and 30 meters and a distance between 0 and 360 degrees from the starting position.

The magic here involves calculating distances and angles with spherical coordinates. The game should be playable on a large (potentially worldly) scale, which means no assuming the Earth is flat. Therefore, from two sets of spherical coordinates (the initial and current position of the AGT device), we need to compute the effective spherical angle and distance between them. Through research, making use of the haversine and bearing equations, where the haversine function calculates the distance between the two positions and the bearing function calculates the angle, seemed like the best approach. They work as follows:

Haversine Distance Function

```
// GET COORDINATE DISTANCE - HAVERSINE FORMULA
double getCoordinateDistance(long La1, long Lo1, long La2, long Lo2) {
    double radius = 6.3781e6; // radius of the Earth (meters)
    // conversion to radians
    double rLa1 = La1*(1E-7) * (PI/180);
    double rLa2 = La2*(1E-7) * (PI/180);
    double rLo1 = Lo1*(1E-7) * (PI/180);
    double rLo2 = Lo2*(1E-7) * (PI/180);

    double a = sin((rLa2-rLa1)/2) * sin((rLa2-rLa1)/2) + cos(rLa1) * cos(rLa2) * sin((rLo2 - rLo1)/2) * sin((rLo2 - rLo1)/2);

    double b = 2 * atan2(sqrt(a), sqrt(1-a));

    return radius * b;
}
```

$$d = 2r \arcsin \left(\sqrt{\frac{1 - \cos(\varphi_2 - \varphi_1) + \cos \varphi_1 \cdot \cos \varphi_2 \cdot (1 - \cos(\lambda_2 - \lambda_1))}{2}} \right)$$

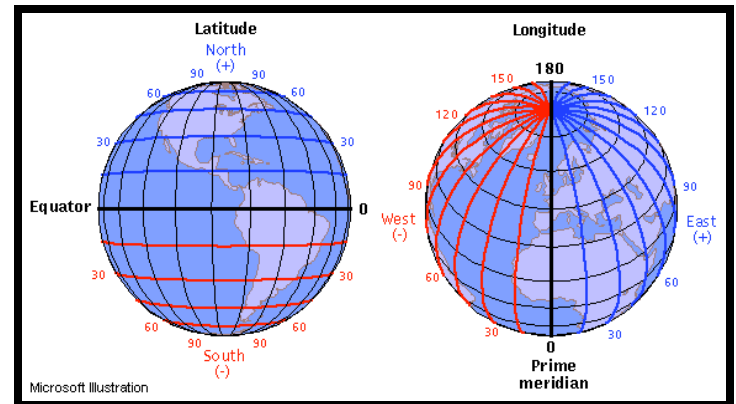
Bearing Angle Function

```
// GET COORDINATE ANGLE (BEARING) HELPER
long getCoordinateBearing(long La1, long Lo1, long La2, long Lo2) {
    double rLa1 = La1*(1E-7) * (PI/180);
    double rLa2 = La2*(1E-7) * (PI/180);
    double rLo1 = Lo1*(1E-7) * (PI/180);
    double rLo2 = Lo2*(1E-7) * (PI/180);

    double y = sin(rLo2-rLo1) * cos(rLa2);
    double x = cos(rLa1) * sin(rLa2) - sin(rLa1)*cos(rLa2)*cos(rLo2-rLo1);

    return (atan2(y, x) * (180/PI)); // returns angle between -180 and 180
}
```

The necessity of these functions becomes apparent once you learn about the system of spherical coordinates. Longitudinal values may work as expected, associated with a vertical position on a sphere. However, Latitudinal distances become difficult to calculate since each longitudinal value forces each horizontal cross-section of the Earth to be a different sized circle.



With these functions, we calculate the current positions angle and distance from the initial position, essentially creating polar coordinates. Once the distance between the target and current polar coordinates reaches a certain threshold (within 2 meters), the treasure has been found! The distance to the target is also made aware to the player visually through an LED meter as shown in the demo. In order to find this distance we use the polar distance formula as shown:

$$d = \sqrt{r_1^2 + r_2^2 - 2 \cdot r_1 r_2 \cdot \cos(\theta_1 - \theta_2)}$$

As the player gets closer to the goal position, the onboard LED blinks faster and faster until it stays lit when the treasure is found. Also, the AGT is writing to the arduino Serial Monitor with the current player positions, the goal positions, and a message demonstrating how close the player is to the treasure (“hotter”, “colder, etc)

The Result

Finally, once the treasure is found, the AGT connects to the iridium satellite network and sends an HTTP POST request with the success message to any destination. The AGT does not need wifi connection for this request: the satellite network does this for you after you make an account with Ground Control and connect the iridium device on the AGT. I made a temporary site on Webhook.site which automatically sets up an endpoint to receive HTTP requests. Here’s an example message after I played the game and won!

POST Request data

```
0000: 59 4f 55 20 46 4f 55 4e 44 20 54 48 45 20 54 52 |YOU FOUND THE TR
0016: 45 41 53 55 52 45 21 21 .. .. .. .. .. |EASURE!!.....
```

The Iridium network can also estimate the approximate location of the message based on the positioning of the satellite themselves.

The Code

Here, we have the main loop function which combines the logic above with functions that interact with the AGT hardware.

```
// MAIN LOOP FUNCTION
void loop() {

  if (WIN) {

    Serial.println(F("YOU FOUND THE TREASURE!!"));
    signalVictory();
    while (true) {
      delay(2000);
      Serial.println(F("YOU FOUND THE TREASURE!!"));
    }

  } else {
    geofenceState currentGeofenceState; // Create storage for the geofence state
    boolean result = myGNSS.getGeofenceState(currentGeofenceState);
    Serial.print(F("getGeofenceState returned: ")); // Print the combined state
    Serial.print(result); // Get the geofence state

    if (!result) // If getGeofenceState did not return true
    {
      Serial.println(F(".")); // Tidy up
      return; // and go round the loop again
    }

    // Print the Geofencing status
    // 0 - Geofencing not available or not reliable; 1 - Geofencing active
    Serial.print(F(". status is: "));
    Serial.print(currentGeofenceState.status);

    // Print the numFences
    Serial.print(F(". numFences is: "));
    Serial.print(currentGeofenceState.numFences);

    // Print the state of the geofence
    // 0 - Unknown; 1 - Inside; 2 - Outside
```

```

Serial.print(F(" The geofence stats is: "));
Serial.print(currentGeofenceState.states[0]);

byte fenceStatus = digitalRead(geofencePin); // Read the geofence pin
digitalWrite(LED, !fenceStatus); // Set the LED (inverted)
Serial.print(F(" Geofence pin (PI014) is: ")); // Print the pin state
Serial.print(fenceStatus);
Serial.println(F("."));

// GET CURRENT POSITION
curr_lat = myGNSS.getLatitude(); // Get the Latitude in degrees * 10^-7
curr_long = myGNSS.getLongitude(); // Get the Longitude in degrees * 10^-7
curr_angle = (getCoordinateBearing(latitude, longitude, curr_lat, curr_long) + 180) %
360;
curr_distance = getCoordinateDistance(latitude, longitude, curr_lat, curr_long);

Serial.print(F("Current Latitude: "));
Serial.print(curr_lat);
Serial.print(F(" Current Longitude: "));
Serial.println(curr_long);

Serial.print(F("Goal Angle: "));
Serial.print(goal_angle);
Serial.print(F(" Goal Distance: "));
Serial.println(goal_distance);

Serial.print(F("Current Angle: "));
Serial.print(curr_angle);
Serial.print(F(" Current Distance: "));
Serial.println(curr_distance);

score = sqrt(curr_distance * curr_distance + goal_distance * goal_distance - 2 *
goal_distance * curr_distance * cos((curr_angle * (2*PI/360)) - (goal_angle * (2*PI/360))));
Serial.print(F("Current Score: "));
Serial.print(score);

if (score < 2) {
    WIN = true;
} if (score < 5) {
    Serial.println(F(": HOT!!"));
} else if (score < 12) {
    Serial.println(F(": Warmerrr..."));
} else if (score < 20) {
    Serial.println(F(": Room Temp"));
} else {
    Serial.println(F(": Ice Cold..."));
}
Serial.println();
}

```

```
digitalWrite(LED, LOW);
delay(score*5);
digitalWrite(LED, HIGH);
delay(score*5);
}
```

Once the player finds the treasure, the victory function runs, which connects the AGT to the Iridium network and sends a victory message.

```
void signalVictory() {
    int signalQuality = -1;
    int err;

    pinMode(LED, OUTPUT); // Make the LED pin an output

    gnssOFF(); // Disable power for the GNSS
    pinMode(gnssBckpBatChgEN, INPUT); // GNSS backup battery charge control
    pinMode(geofencePin, INPUT); // Configure the geofence pin as an input
    pinMode(iridiumPwrEN, OUTPUT); // Configure the Iridium Power Pin
    digitalWrite(iridiumPwrEN, LOW); // Disable Iridium Power
    pinMode(superCapChgEN, OUTPUT); // Configure the super capacitor charger enable pin
    digitalWrite(superCapChgEN, LOW); // Disable the super capacitor charger
    pinMode(iridiumSleep, OUTPUT); // Iridium 9603N On/Off (Sleep) pin
    digitalWrite(iridiumSleep, LOW); // Put the Iridium 9603N to sleep
    pinMode(iridiumRI, INPUT); // Configure the Iridium Ring Indicator as an input
    pinMode(iridiumNA, INPUT); // Configure the Iridium Network Available as an input
    pinMode(superCapPGOOD, INPUT); // Configure the super capacitor charger PGOOD input

    modem.endSerialPort();

    // Enable the supercapacitor
    Serial.println(F("Enabling the supercapacitor charger..."));
    digitalWrite(superCapChgEN, HIGH);
    delay(1000);
    // wait for the capacitor to charge
    while (digitalRead(superCapPGOOD) == false)
    {
        Serial.println(F("Waiting for supercapacitors to charge..."));
        delay(1000);
    }
    Serial.println(F("Supercapacitors charged!"));

    // Enable power for the 9603N (Iridium Connection)
    Serial.println(F("Enabling 9603N power..."));
    digitalWrite(iridiumPwrEN, HIGH); // Enable Iridium Power
    delay(1000);
    modem.setPowerProfile(IridiumSBD::USB_POWER_PROFILE);

    // Startup the modem
    Serial.println(F("Starting modem..."));
}
```

```

err = modem.begin();
if (err != ISBD_SUCCESS)
{
    Serial.print(F("Begin failed: error "));
    Serial.println(err);
    if (err == ISBD_NO_MODEM_DETECTED)
        Serial.println(F("No modem detected: check wiring."));
    return;
}

// Test the signal quality.
err = modem.getSignalQuality(signalQuality);
if (err != ISBD_SUCCESS)
{
    Serial.print(F("SignalQuality failed: error "));
    Serial.println(err);
    return;
}
modem.useMSSTMWorkaround(false);

Serial.print(F("On a scale of 0 to 5, signal quality is currently "));
Serial.print(signalQuality);
Serial.println(F("."));

// Send the message
Serial.println(F("Trying to send the message. This might take several minutes."));
err = modem.sendSBDText("YOU FOUND THE TREASURE!!");
if (err != ISBD_SUCCESS)
{
    Serial.print(F("sendSBDText failed: error "));
    Serial.println(err);
    if (err == ISBD_SENDRECEIVE_TIMEOUT)
        Serial.println(F("Try again with a better view of the sky."));
}

// POWER DOWN IRIDIUM CONNECTION

// Clear the Mobile Originated message buffer
Serial.println(F("Clearing the MO buffer."));
err = modem.clearBuffers(ISBD_CLEAR_MO); // Clear MO buffer
if (err != ISBD_SUCCESS)
{
    Serial.print(F("clearBuffers failed: error "));
    Serial.println(err);
}

// Power down the modem
Serial.println(F("Putting the 9603N to sleep."));
err = modem.sleep();
if (err != ISBD_SUCCESS)
{
    Serial.print(F("sleep failed: error "));

```



```
        Serial.println(err);
    }

    Serial.println(F("Disabling 9603N power..."));
    digitalWrite(iridiumPwrEN, LOW); // Disable Iridium Power

    Serial.println(F("Disabling the supercapacitor charger..."));
    digitalWrite(superCapChgEN, LOW); // Disable the super capacitor charger

    Serial.println(F("Game Complete!"));
}
```

Next Steps

All of these components together creates a fun hide and seek game with your Artemis Global Tracker! There are many ways one could extend this example and create a higher functioning game. For one, you could create a website designed to take a POST request after a player wins the game and show stats from the game (ex: time until find, score, etc.). Also, you could add hardware components along with the AGT to graphically show game stats for the user while playing. For example, an LED meter would be a great way to show live score updates to the player.

Credits

This project could not have happened without the documentation and examples from the Sparkfun AGT examples repository [found here](#). Some of Sparkfun's setup functions and definitions were necessary in the working game, and can be found in the resulting code.